(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : 0 583 117 A2

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number : 93306031.1

(22) Date of filing : 30.07.93

(51) Int. Cl.^{5} : G06F 9/44

(30) Priority : 12.08.92 JP 215032/92

(43) Date of publication of application : 16.02.94 Bulletin 94/07

(84) Designated Contracting States : DE FR GB

(71) Applicant : International Business Machines Corporation
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : Ono, Shigeichi
1-14-13, Sakushin-dai
Chiba-shi , Chiba-ken (JP)

(74) Representative : Burt, Roger James, Dr.
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) **A device and method for converting computer programming languages.**

(57) A device and method for converting programming languages that can be applied to program conversion between many computer programming languages. When model statements in an origin language, model statements in a target language, and conversion rules for word (variable) sections are inputted from the screen of a conversion rule input means, a conversion rule generation means generates a conversion rule according to this input and stores it in the conversion rule database. The conversion rule is composed of a search key, a word conversion direction section, and a target language generation program. The search key is used when a conversion rule needed for the conversion of an unconverted origin program is searched for in a database and fetched, the word conversion direction section is used for calling the word conversion table or the word conversion program, and the target language generation program is used for generating a converted target program. The conversion rule input means, conversion rule generation means, and conversion rule drive means do not depend on the origin language and target language of the languages involved in conversion.

EP 0 583 117 A2

Jouve, 18, rue Saint-Denis, 75001 PARIS

SPECIFICATION

This invention relates to a device and method for converting computer programming languages, and particularly, to a highly versatile device and method for converting computer programming languages that can be used for converting programs between a number of computer programming languages without being limiting to conversion of programs between two specific programming languages.

A program that gives a job procedure to a computer is written in computer programming language (programming language). There are many computer programming languages. When viewed in terms of language level, there are languages that are directly dependent on the computer, which are called machine languages, and languages that are much closer to human languages, which are called high-level languages. When viewed in terms of language application, there is FORTRAN, which is a scientific computation-oriented language and COBOL, which is a business-oriented language. Some languages are developed for specific computers by specific laboratories or enterprises. In recent years, there have been general-user-oriented languages, called 4th generation languages, that are closer to the applications. In addition, there may be several different dialects because of differences in level, even in the same types of language. There are many cases where a program written in a computer programming language (e.g., FORTRAN) must be converted into a program written in another computer programming language (e.g., COBOL). This situation arises when it is desired to run a program written in a language for a certain computer on another computer and there is a need to rewrite that program into a language for the other different computer.

Therefore, a device for converting computer programming languages that converts programs between different computer languages (hereinafter, the unconverted language is called the origin language and the converted language is called the target language) is needed. Here, conversion means to change the program from one language (origin language) into another language (target language).

Many computer programming language converters have been- proposed in the past. The problem of the conventional computer programming language converter is that it is necessary to modify the built-in converter program even if only the level of the unconverted origin language is changed, because the conversion method between the different languages in the converter built-in program is fixed. Furthermore, it is necessary to closely investigate the differences between the unconverted origin language and the converted target language at the stage of making a converter. It takes a tremendous amount of time to make a programming language converter because all the specifications will be different in conversion between different languages, for example, between FORTRAN and COBOL, or between application languages called 4th generation language. Furthermore, engineers who know both the unconverted origin language and the converted target language well are needed to make a programming language converter.

A programming language translator, for example, such as is disclosed in PUPA No. 2-10434, has been proposed in the past to solve this kind of problem. This provides a comparison table of basic syntax (instruction statements) between origin language and target language, for analyzing the unconverted program syntax, and for selecting an appropriate basic syntax for the target language from this table. In the translator disclosed in PUPA No. 2-10434, making use of a new origin language and the target language only requires storing that new basic syntax in the table. However, the basic syntax in this table stores sections expressing those function, for example, the GO TO section (hereafter called key words), and variable sections, for example, a %1 section (hereafter called word) together. For the conversion into the target language in the corresponding table, the key words of the basic syntax and words are converted together. Therefore, in this conventional translator example, since key words and words are dealt with as one set and converted together, it is impossible to convert words themselves according to independent rules. This means it lacks versatility as a programming language translator.

In PUPA No. 2-183339, another conventional example, a method to obtain versatility in a computer programming language converter by incorporating lexical analysis and syntax analysis as independent routines not based on the origin language and target language is proposed. However, there is also no suggestion in this conventional example about a method to materialize a computer programming language converter with higher versatility by separating the instruction statements into a key word section expressing functions and a word section expressing variables for conversion, and by carrying out these conversions independently.

Therefore, the object of this invention is to provide a device and method for converting computer programming languages that have much higher versatility than conventional converters, and which can convert programs between many more computer programming languages. According to one embodiment of the present invention, conversion rules are generated automatically when an operator inputs a conversion model on the computer input screen that converts from a model statement of an origin language to a model statement of a target language. At this time, a device and method for converting computer programming languages which have high versatility,

which can give instructions to convert key words expressing functions and words expressing variables in the instruction statements according to separate rules are provided.

This device for converting computer programming languages of the invention has a conversion rule input means, a conversion rule generation means, and a conversion rule drive means. The conversion rule input means has an origin language input section that describes and inputs modelled instruction statements in the origin language in which the unconverted program is written according to the syntax of the origin language, a target language input section that describes and inputs modelled instruction statements in the post-conversion target language, which perform the functions of the instruction statements in the origin language, and a word conversion input section that describes and inputs the rules to convert individual variables, that is, words, that form the instruction statements in the origin language into variables of the instruction statements in the target language.

The conversion rule generation means generates a conversion rule search key by using the input model statements that were inputted in the origin language section, generates a word conversion table or a word conversion program call function by using the rules that were inputted in the word conversion input section, generates a target language generation program by using the output model statements that were inputted in the target language input section, and generates a conversion rule taking these as a set. This conversion rule is stored in the conversion rule database.

When an unconverted origin program is input, the conversion rule drive means analyzes the origin program and fetches conversion rules to convert the instruction statements in the origin language from the conversion rule database by using a conversion rule search key. The conversion rule drive means further converts the words of the unconverted instruction statements into the words of the instruction statements of the converted program by using a word conversion table call function that is contained in the fetched conversion rule, and further, generates and outputs instruction statements in the converted target language by using a target language generation program that is also contained in the conversion rule.

According to this invention, the language conversion of programs can be executed by inputting origin language model statements, target language model statements, and word (variable) section conversion rules using the conversion rule input means, by creating conversion rules by the conversion rule generation means based on this input, and by storing them in the conversion rule database. In this way, the model statements in the origin language at the origin section can be described by any programmer of the origin language and this can be inputted by the conversion rule input means. In addition, programmers of the target language can describe it in the model statements of the target language through an understanding of the functions of the model statements, and this can be inputted together with the conversion rules for the word sections by the conversion rule input means. Therefore, conversion rules able to convert the word sections independently and conversion rules for the instruction statements can be generated at the same time. In addition, it is not necessary to generate all the conversion rules required for conversion from an origin language into a target language at one time. It can be made so that conversion rules that are needed when converting from an origin program into a target program are generated as occasion demands. Therefore, when a certain number of programs are converted, the conversion rules needed for conversion from an origin language into a target language are stored in the conversion rule database.

According to this invention, conversion is materialized when the conversion rule drive means analyzes a program written in an origin language for each instruction or for each instruction group, collates these instructions or instruction groups with the model statements in the origin language of the conversion rules stored in the conversion rule database, searches for ones that agree, and takes out programs to create model statements in a target language corresponding to those instruction statements or instruction groups. At this time, the table or program call function to convert the word sections is also taken out similarly from the conversion rule, and words are converted by using the word conversion table.

According to this invention, in principle, the conversion rule input means, conversion rule generation means, and conversion rule drive means do not depend on the types of the origin languages and target languages. Therefore, the versatility of this device of the invention for converting programming languages is very high.

According to this invention, programming languages can be converted by inputting model statements in an origin language, model statements in a target language, and conversion rules for the word (variable) sections by using a conversion rule input means, by establishing conversion rules by using a conversion rule generation means according to this input, and by storing them in a conversion rule database. In this way, any programmer of the origin language can describe model statements in the origin language at the origin section and input these by using a conversion rule input means. In addition, if the functions of these model statements are understood, programmers of the target language can describe them in model statements in the target language, and these can be inputted by a conversion rule input means together with the conversion rules for the word sections. Therefore, conversion rules in which

the word sections can be converted independently can be generated simultaneously with the conversion rules for the instruction statements. In addition, as understood from the above explanation, a conversion rule input means, a conversion rule generation means, and a conversion rule drive means, which are the main sections of a programming language converter of this invention, do not depend on the types of conversion languages involved. Therefore, a programming language converter of high versatility can be materialized. Since the variables of the programming languages can be converted separately from the key words expressing functions according to the device and method for converting programming languages of this invention, its versatility can be increased. Furthermore, since the device and method for converting programming languages of this invention stores part of the conversion rules as independent files for maintenance, it is superior in terms of conversion rule maintenance.

An embodiment of the present invention will now be described with reference to the accompanying drawings in which:

Figure 1 is a schematic block diagram showing a device for converting computer programming languages according to an embodiment of this invention;

Figure 2 is a schematic block diagram showing the generation and use of the conversion rules of this embodiment;

Figure 3 is a drawing showing the conversion rule input screen of the conversion rule input means of this embodiment;

Figure 4 is a flowchart showing the conversion rule generation procedure of this embodiment (part 1);

Figure 5 is a flowchart showing the conversion rule generation procedure of this embodiment (part 2);

Figure 6 is a schematic block diagram showing the files associated with the conversion rules;

Figure 7 is a flowchart showing the programming language conversion procedure;

Figure 8 is a drawing showing an example of an unconverted origin program;

Figure 9 is a drawing showing an example of a converted target program;

Figure 10 is a drawing showing an example of a conversion rule input screen;

Figure 11 is a drawing showing an example of a conversion rule input screen;

Figure 12 is a drawing showing an example of a conversion rule input screen;

Figure 13 is a drawing showing an example of a conversion rule input screen;

Figure 14 is a drawing showing an example of a conversion rule input screen;

Figure 15 is a drawing showing an example of a conversion rule;

Figure 16 is a drawing showing an example of a conversion rule;

Figure 17 is a drawing showing an example of a conversion rule;

Figure 18 is a drawing showing an example of a conversion rule;

Figure 19 is a drawing showing an example of a conversion rule;

Figure 20 is a drawing showing word conversion tables and word conversion logic;

Figure 21 is a drawing showing an intermediate file of an unconverted origin program;

Figure 22 is a drawing showing a conversion rule drive program of a conversion rule drive means; and

Figure 23 is a flowchart showing the collation and fetch procedures of a plurality of conversion unit statements and conversion rule.

Figure 1 is a block diagram illustrating a schematic configuration of general-purpose computer programming language converter 1, which is an embodiment of the present invention. Converter 1 has a conversion rule generation means 2. This conversion rule generation means 2 is connected to a conversion rule input means 3, and generates conversion rules according to the input from conversion rule input means 3. The generated conversion rules are stored in a conversion rule database 4. A word conversion table creation means 5 creates a word conversion table 7 according to the input from a word conversion table input means 6. An origin language program input means 8 inputs an unconverted origin program 80 and carries out preprocessing before conversion, such as analysis. A conversion rule drive means 9 searches conversion rules database 4 for conversion rules corresponding to the input origin program, fetches them, and converts into a target program 90 according to the conversion rules. In addition, conversion rule drive means 9 calls word conversion table 7 according to the same conversion rules as mentioned above and carries out word conversion between an origin program 80 and the target program 90. The converted program is output from a target language program output section 10 and a target program 90 that has been converted into target language is formed.

Next, the generation of conversion rules that is used in this converter 1 will be explained below. As shown in Figure 2, the conversion rule generation means 2 is connected to a user interface that provides an input screen 30 for the conversion rule input means 3, a conversion rule text image file 22, a conversion rule search key file 23, and a conversion rule database 4. A conversion rule drive means 9 fetches conversion rules by accessing a conversion rule database 4 by using conversion rule search key file 23 when origin program 80 is converted into target pro-

gram 90. Conversion rules are generated before conversion from the origin program 80 into the target program 90 is carried out, but necessary conversion rule can also be generated during conversion. This is done by calling the conversion rule generation means 2 from the conversion rule drive means 9 at times when there is no appropriate conversion rule during conversion, as explained later. The conversion rule text image file 22 is used for maintenance of conversion rules, to be explained later.

The screen 30 of the conversion rule input means 3 provides an initialization input screen and a conversion rule input screen. The initialization input screen specifies an origin language subject to conversion, and a target language. For example, conversion from COBOL for system A into COBOL for system B, or conversion from FORTRAN to COBOL, etc. Figure 3 shows the conversion rule input screen 30. Operators can also input, modify, or give directions, such as delete or change, from this computer input screen when a rule is established. This computer input screen 30 has, from the top, an origin language input section 31, a word conversion input section 32, and a target language input section 33, and at the lower section of the screen, program function keys 34 that direct the after-input process, such as to create a new conversion rule, or modify existing rule, or add, delete, change, or replace a rule. The program accompanying this screen 30 is also provided with a function to edit inputted text.

A rule inputted on conversion rule input screen 30 is stored in the conversion rule text image file 22 in a form similar to its input form. This is done so that the conversion rule can be easily displayed again on the screen 30 for later maintenance of a conversion rule.

A conversion rule search key is stored in conversion rule search key file 23. This search key is converted so that the origin language sections of the inputted conversion rules can be easily searched with the programming language used in this embodiment, PROLOG. In this embodiment, this converted origin language section is called the internal form for searching.

As an example, generation of a conversion rule to convert an IF statement, an instruction statement in FORTRAN, into an instruction statement having the corresponding function in COBOL will be explained below. Figures 4 and 5 are flowcharts showing the conversion rule generation procedures of this embodiment. In initialization (block 41) in Figure 4, the types of language subject to conversion, that is, the origin language (FORTRAN) and the target language (COBOL) are specified on the initialization screen. In addition, origin language specific delimiters and the like on description are specified as the need arises.

In screen input (block 42), operators input FORTRAN and COBOL model statements and the like, which will be the basis for the establishment of conversion rules, on the computer input screen 30 of the conversion rule input means 3. The instruction statements in the origin language to be inputted by operators in the origin language input section 31 of screen 30 in Figure 3 are modelled as follows. That is, instruction words that form instruction statements in the origin language, that is, words expressing functions, are displayed as they are. In the instruction statement, the portions corresponding to variables are expressed by character strings or character to which an asterisk (*) is attached at the head.

The general form of an IF statements in FORTRAN is IF(A) N1, N2, N3. A stands for a relational expression. This IF statement means that statement number N1, N2, or N3 will be executed, according to the value of A (positive, zero, or negative). Assume that the statement numbers N1, N2, and N3 are 100, 200, and 300 respectively. If A<0, the IF statement gives an instruction to execute statement number 100. If A=0, the IF statement gives an instruction to execute statement number 200. And if A>0, the IF statement gives an instruction to execute statement number 300. In IF statements, A, N1, N2, and N3 correspond to variables expressed in different forms respectively, according to the content of the program. In this invention, these variables are called word sections. In IF statements, IF corresponds to an instruction word expressing the function of an instruction statement, and has a fixed form in the programs in the same language. Therefore, in this invention, the instruction word IF will be a key word sections of a conversion rule.

In the origin language input section 31 on the screen 30 in Figure 30, a statement in which an IF statement is modelled, that is, IF(*A) *N1, *N2, *N3 is inputted, in conformance with FORTRAN grammar. That is, the IF portion of the instruction word is used as it is as a key word. The word sections are expressed by characters or character strings in which an asterisk (*) is given at the head. In this way, an appropriate conversion rule can be searched for by collating with the conversion rule search key that is generated from this model statement for each conversion unit statement of the origin language program to be converted by using the unification function of PROLOG language. In this way, the expression of instruction statements in the origin language in a form to be collated by a conversion rule search key is called modelling.

When a FORTRAN instruction statement, IF(A) 100, 200, 300, is described in a COBOL instruction statement having the same function as in FORTRAN, it will be IF A<0 PERFORM L100, IF A=0 PERFORM L200, IF A>0 PERFORM L300. The statement numbers in FORTRAN are expressed by numerals, but they correspond to labels in COBOL. Here, the COBOL label is a character string starting with a charac-

ter from the alphabet.

In target language input section 33, instruction statements are described in the target language so as to be able to realize a function or an intended function (it might be desired to convert into a different function in some cases) that is the same as an instruction statement inputted in the origin language input section 31. To execute a function that is equal to the modelled instruction statement of FORTRAN, IF(*A) *N1, *N2, *N3, mentioned above in COBOL, the statements are described as follows in the target language input section 33 on screen 30.

IF *A<0 PERFORM *XL1.
IF *A=0 PERFORM *XL2.
IF *A>0 PERFORM *XL3.

Here, *XL1, *XL2, and *XL3 are the modelled forms of element names that are created by converter 1 according to a conversion direction that is inputted in word conversion input section 32, to be explained later, and that can be used in COBOL.

The following inputting is carried out in word conversion input section 32. If the languages involved in conversion, the origin language and target language, have different systems, the language components to be used are different. For example, the statement numbers in FORTRAN are expressed by numerals, but they correspond to labels in COBOL. The labels in COBOL are normally character strings starting with an alphabetic character. In addition, even if the languages are of the same system, reserved words expressing device names may differ. Therefore, the purpose of inputting into the word conversion input section 32 is for generating a PROLOG language predicate for calling the word conversion program or the word conversion table from the inputted information.

For example, a direction to word conversion input section 32, such as;

*N1 Label *XL1

means to convert a statement number in FORTRAN into a label with an alphabetic character in COBOL. This is, for example, transformed into a form called an "item," such as;

& table (Label, N1, XL1)

in PROLOG language by a conversion rule generation means 2 described later. Here, when, for example, a statement number 100 is given to N1, a label in which the alphabetic character L is given at the head of 100, such as L100, is generated for XL1. Therefore, the following is inputted to word conversion input section 32.

*N1 Label *XL1
*N2 Label *XL2
*N3 Label *XL3

Next, whether to establish a new conversion rule using the program function key 34 or to use the existing conversion rule by modifying it is directed on the screen 30 in Figure 3. If the existing conversion rule is used by modifying it, the conversion rule search key file 23 is searched for the same conversion rule as that inputted on the screen 30 in Figure 3 or a similar conversion rule (block 43), and an appropriate conversion rule is read from the conversion rule text image file 22 and displayed on the screen 30 (block 44).

In screen operation (block 45), operators create, modify, and edit the conversion rule on the screen 30 in Figure 3, and direct measures for the later by program function key 34. These directions include add, delete, change, and replace.

The contents inputted into the origin language input section 31, word conversion input section 32, and target language input section 33 on the input screen 30 of the conversion model input means in Figure 3 are processed by the conversion rule generation means 2 to generate a conversion rule.

First, in the lexical analysis of the origin language (block 46), the model statements in the origin language that are inputted into the origin language input section 31 are decomposed into variable and key word elements while referring to the delimiter table 47 by language or by instruction statement that is specified on the initialization screen (block 41).

In conversion into the internal form for searching (block 48), the elements that have already been decomposed are converted into an internal form for searching so that they can later be used as search keys in the conversion rule drive means 9. That is, the model statements in the origin language are converted into keys that will be used for searching for conversion rules in later conversion of the origin program. At this time, the key creation method can be changed so that it matches the search method in conversion rule drive means 9. For example, even if the word orders of key words used are different, if it is desired to treat them as the same key, flexible collation can be performed by giving key words that are rearranged in ascending order or in descending order as a sub-key.

For example, a model statement that is inputted in the origin language input section 31, such as;

IF(*A) *N1, *N2, *N3

is converted into an internal form for searching by conversion rule generation means 2, such as;

convert (["IF", "(", A, ")", N1, ",", N2, ",", N3])

This internal form for searching, expressed by "convert (...)" is placed in the first line of the conversion rule so that it can be used as a conversion rule search key. When the conversion rule drive means 9 searches for and fetches a necessary conversion rule, this search key is used for the purpose of collating whether the conversion rule is necessary or not.

In the process of word conversion input section 32 (block 49), the following process is carried out. If the languages involved in conversion have different language systems, the language components to be used are different. Therefore, in the process of word conversion input section 32, a PROLOG language

predicate for calling a word conversion program that performs word conversion or a word conversion table is generated from the inputted information.

For example, a direction to word conversion input section 32, such as;

*N1 Label *XL1

means to convert a statement number in FORTRAN into a label with an alphabetic character in COBOL. This is, for example, transformed into a form called an "item," such as;

& table (label, N1, XL1)

in PROLOG language. Here, when, for example, statement number 100 is given to N1, a label in which the alphabetic character L is given at the head of the number 100, such as L100, is generated for XL1.

Therefore, the word conversion input section 32 is converted into PROLOG language predicates as a word conversion direction section for calling a word conversion program or a word conversion table, such as the following.

```
<- table (label, N1, XL1)
& table (label, N2, XL2)
& table (label, N3, XL3)
```

These lines are placed next to the conversion rule search key expressed by "convert (...)" that is mentioned above in the conversion rule.

In the process of the target language input section (block 51), the model statements that are inputted in the target language input section 33 are analyzed lexically and converted into a form enabling generation of the target language when conversion rules are taken out and processed in the conversion rule drive means 9. At this time, since the variables used in the target language section are defined in the origin language section or in the word conversion section; error checking is also carried out so that improper things are not generated in generating the target language due to variables not yet defined.

For example,

IF *A<0 PERFORM *XL1.

mentioned above is converted into the form of

&Z1:= 'IF' | |A| | '<0 PERFORM' | |XL1| | '.'

This means to generate

IF X<0 PERFORM L100.

by jointly operating the information (*A is X) obtained from the character string enclosed by quotation marks and the origin language section and the information (XL1 is L100) obtained from the word conversion section at the time of execution by the conversion rule drive means. Therefore, in the process of the target language section, a target language generation program such as the following is generated by the conversion rule generation means 2 and placed next to the word conversion direction section.

```
&Z1:= 'IF' | |A| |' <0 PERFORM' | |XL1| | '.'
&Z2:= 'IF' | |A| |' <0 PERFORM' | |XL2| | '.'
&Z3:= 'IF' | |A| |' <0 PERFORM' | |XL3| | '.'
&prst (Z1, out) & n1 (out)
&prst (Z2, out) & n1 (out)
&prst (Z3, out) & n1 (out)
```

As explained above, a conversion rule search key created from the input of the origin language input section 31, a word conversion direction section created from the input of word conversion input section 32, and a target language generation program created from the input of the target language input section 33 as a set constitute conversion rule 99, such as shown in Figure 15. This conversion rule 99 converts an IF statement in FORTRAN into an instruction statement in COBOL having the same function.

If there is an error in the process mentioned above, the section with the error is indicated to the operator and correction is required (block 52).

When there are no more errors on the conversion rule establishment screen, the operator gives a direction, such as add, change, and replace the conversion rule 99 (block 53). The processing at this time will be explained by referring to Figures 5, 6, and 15 together. If a direction for addition is given, it is confirmed that the same conversion rule 99 is not already in the conversion rule search key file 23 (block 54). Next, the conversion rule search key, conversion rule search sub-key, pointer to conversion rule text image file 22, and pointer to conversion rule database 4, which are created by the above process, are stored in the conversion rule search key file 23 (block 55). Then, a conversion rule (a set consisting of a conversion rule search key, word conversion direction section, and target language generation section) 99 is stored in the conversion rule database 4 (block 56). In this way, a conversion rule database 4 is constructed. Next, the contents of the conversion rule input screen 30 at the time when this conversion rule 99 is established are stored in the conversion rule text image file 22 so that it can be easily displayed again on the screen for later maintenance of the conversion rule 99 (block 57).

If a direction for change is given, it is confirmed that the conversion rule search key of the conversion rule 99 is unchanged (block 58). The section other than the conversion rule search key of the applicable conversion rule in the conversion rule database 4 is changed into a new conversion rule (block 59). Then, the section other than the origin language input section in the text image file 22 is changed (block 60).

If a direction for replacement is given, it is confirmed that there is no new conversion rule search key for the conversion rule 99 is in the key file 23 (block 61). The old search key in the key file 23 is deleted and a new search key is registered (block 62). All the applicable old conversion rules in the conversion rule database 4 are replaced with new ones (block 63). Then, the applicable old section in the text image file 22 is replaced with a new one (block 64). If there is another conversion rule to be established, the same processing is repeated (block 65). If there

is no more conversion rule establishment, the process ends (block 66).

In the conversion rule establishment explained above, the associated files, as shown in Figure 6, are created centered on the conversion rule search key that is generated by analyzing the model statements in the origin language. That is, the origin language model statement 31 will be a conversion rule search key. The conversion rule database 4, the conversion rule search key file 23, and the conversion rule text image file 22 are constructed centered on this key.

The conversion rule generation mentioned above is briefly explained below. Figure 8 shows some statements of an unconverted origin program 80. Statement 81 in the program is an IF statement in FORTRAN as mentioned above. Figure 9 shows some corresponding statements of a converted target program. Statements 91 are instruction statements in COBOL, which are equivalent to that in FORTRAN, into which the IF statement 81 in FORTRAN is converted. Figure 10 shows an example of word conversion rules and model statements of the origin language and target language that are inputted on conversion rule input screen 30 by the operator when a conversion rule to convert the origin program 81 in Figure 8 into instruction statements 91 in COBOL having a corresponding function in the target program 90 in Figure 9. Figure 15 shows the conversion rule 99 that is generated from the input in Figure 10 by the conversion rule generation means 2 to convert the IF statement 81 in FORTRAN mentioned above into instruction statements 91 in COBOL having the same function. The conversion rule 99 consists of a set of a conversion rule search key that is generated from the origin language input section, a word conversion direction section that is generated from the word conversion input section, and a target language generation program that is generated from the target language input section. This conversion rule 99 is stored in the conversion rule database 4.

Next, a method to convert the origin program 80 into a target program 90 using the conversion rule 99 that is generated in this way is explained by referring to Figure 7.

As shown in Figure 7, when an origin program 80 is converted into a target program 90, operation environment parameters 71a are set up in the initialization process and the types of languages that are involved in conversion are specified (block 71). In origin language program input, the origin language program 80 for each record is read from the input device specified in the initialization process (block 72). If a conversion unit statement in the origin language program 80 is divided into two records or more, a plurality of records are reconstructed in the conversion unit statement in this origin language program input 8. Comment lines or comments in the origin program 80 are stored separately for later output. Then, the conversion unit statement is transferred to the following processing.

Next, lexical analysis of the conversion unit statement is carried out (block 73). In lexical analysis, each conversion unit is decomposed into components by referring to language delimiter table 47 by using the same lexical analysis feature as that in creating a conversion rule explained in Figure 4. For example, in instruction statement 81, an IF statement in FORTRAN in the origin program 80 shown in Figure 8;

IF(X) 100, 200, 300

The following components are decomposed and analyzed.

[IF], [(], [X], [)], [100], [,], [200], [,], [300]

In the following internal form conversion for collation (block 74), these decomposed components are converted into an internal form so that they can be easily collated with the conversion rule search key (Refer to the first line in Figure 15), and are stored in intermediate file 21 or in a storage device. For example, in the case of the IF statement mentioned above, the intermediate file 21 has a form shown in line 21a in Figure 21. The lexical analysis feature (block 73) and the internal form conversion feature for collation (block 74) repeat the operation mentioned above until the reading of one file of origin program 80 is completed (block 75).

In the conversion assistance information extract support feature (block 76), language conversion can be carried out sequentially from the initial record in many cases, but if the information before and after an instruction statement in the conversion process can be referred to, the conversion efficiency can be increased. Here, it is made possible to fetch information from a conversion rule collation key that is stored in intermediate file 21 that has been created in advance. For example, this may be information on variables of components that are used for arithmetic operation or character operation. Since the needed information varies depending on the languages involved in conversion, an interface to conversion assistance information file 77 is provided.

The origin language program fetch feature (block 78) fetches the origin program 80 that has been converted from the intermediate file 21 into conversion rule collation key form (line 21a in Figure 21). At this time, the conversion unit statements for the numbers given to the characteristic information of conversion rule database 4 are fetched and transferred to the following conversion rule database collation feature (block 79).

In the conversion rule database collation feature (block 79), an origin program 80 that is stored in intermediate file 21 is converted into conversion rule collation key form (line 21a in Figure 21), which is basically formed of components. A search key (the first line in Figure 15) of a conversion rule 99, which is stored in conversion rule database 4 and conversion rule

search key file 23, is generated by conversion rule generation means 2 and is formed of key words and variables. For the collation of these, among the components, conversion rule database 4 is searched for a conversion rule search key having the same key words and arranged in the same order and having the same number of variables as that of the components other than the key words. If the positional relation between the key words and the component other than them is the same, it is considered that an appropriate rule exists and the conversion is started according to that conversion rule. The search for this conversion rule search key is carried out by using the search key in the key file 23. And if an appropriate search key is found, the corresponding conversion rule 99 is fetched by using the pointer to a corresponding conversion rule 99 in conversion rule database 4 that is stored in the key file 23 (Refer to Figure 6). When a conversion rule is searched for, a plurality of conversion unit statements are collated in some cases. The flowchart of this is as shown in Figure 23.

If an appropriate conversion rule does not exist in the conversion rule database 4 (G in block 101), a message is issued and the creation of a new conversion rule for this is requested of the operator (block 102). The operator gives a direction as to whether to establish a new rule or to carry out establishment later (block 103). If a new rule is established, conversion is carried out according to this rule (F of block 103). If a rule is not established, a message indicating that the conversion could not be carried out is issued and the operator goes to the next process (block 104).

When an appropriate conversion rule is found (F of block 101), the conversion rule 99 is fetched from the conversion rule database 4, the word conversion direction section and the target language generation program are incidentally fetched from the conversion rule 99 (Refer to Figure 15) (block 105), and word conversion and target language generation are carried out. The execution of target language generation is first carried out from the calling of the word conversion library 7 and conversion of words even though this depends on the specification of conversion rule 99 (block 107). The variable section of the target language generation program is replaced with converted words (block 108). The target program 90 of the target language is generated by the target language generation program (block 109) and this is outputted to the specified output device, from the target language output section 10. At this time, comment lines or comments that were separated out at the time of input and stored separately are called and outputted when the need arises.

If the origin program (IF statement in FORTRAN, 81 in Figure 8) of this embodiment is converted into COBOL (target program 91 in Figure 9), the word conversion table 7 in Figure 20 stored in the word conversion library 7 is fetched from the word conversion direction section of the conversion rule 99 (Figure 15) that is fetched in block 105. Statement number 100 in FORTRAN is converted to label L100 in COBOL (blocks 107 and 108). Then, the target language generation program of conversion rule 99 (Figure 15) generates the instruction statements 91 of the target program in Figure 9 by jointly operating the information ("A is X) obtained from the character strings enclosed with quotation marks (' ') and the origin language section and the information (XL1 is L100, etc.) obtained from the word conversion section.

The word conversion table and word conversion logic 7 shown in Figure 20 are such that they are to be created and registered from the word conversion table input 6 by word conversion table creation means 5 in the origin language to be converted and the target language separately, as the need arises.

In this way, the conversion of one origin program 80 ends (block 110), and if there is no other conversion of origin program (block 111) to be performed, the program language conversion process ends (block 112).

Figure 22 shows the main part of an example of a conversion rule drive program 100 of the conversion rule drive means 9 for the actual execution of the program language conversion process shown in Figure 7 explained above. This conversion rule drive program 100 fetches the internal form 21a of the origin program 81 that is stored in the intermediate file 21 in Figure 21 sequentially with

data (M, List)

calls an appropriate conversion rule 99 (Figure 15) with

convert (List)

and generates a target program 91 shown in Figure 9 as mentioned above. As understood from the above, this conversion rule drive program 100 has a form that does not depend on either the origin language or the target language of the languages involved in conversion.

Next, another conversion example according to this embodiment is explained by taking the origin program 80 shown in Figure 8 as an example.

There is a desire in some cases to convert a plurality of instruction statements of an origin program as a set into a target program. For example, when the character strings defined as;

DATA HEAD(1)/'HEAD1'/
DATA HEAD(2)/'HEAD2'/
DATA HEAD(3)/'HEAD3'/

in FORTRAN are moved to another different area TITLE, they are described like the following one set of instruction statements shown in origin program 82 in Figure 8;

TITLE(1) = HEAD(1)
TITLE(2) = HEAD(2)
TITLE(3) = HEAD(3)

When this is converted into COBOL, a conver-

sion rule for converting them as if they are one instruction statement like;

MOVE HEAD TO TITLE

shown in target program 92 in Figure 9 is established as follows.

In cases when a model statement is inputted into the conversion rule input screen 30 and it is desired to convert a plurality of instruction statements as a set, if an exclamation mark (!) is attached to the head of each model instruction statement of the origin language input section as shown in Figure 11, these can be dealt with as a group. That is, they are described as follows:

Origin language input section

!*TO(*I) = *FROM(*I)
!*TO(*J) = *FROM(*J)
!*TO(*K) = *FROM(*K)

Target language input section

MOVE *COBOLY TO *COBOLX

These are converted and stored into the conversion rule search key of the conversion rule 99 shown in Figure 16 in such a form that can be collated with a plurality of statements in the origin language by the conversion rule generation means 2.

In the word conversion input section, in an example when the following instruction statements in FORTRAN mentioned above;

!*TO(*I) = *FROM(*I)
!*TO(*J) = *FROM(*J)
!*TO(*K) = *FROM(*K)

are converted into a COBOL instruction statement as follows;

MOVE *COBOLY TO *COBOLX

the *I, *J, and *K values must be in succession. In such a case, for example, the following;

*I relation *J
*J relation *K
*TO name *COBOLX
*FROM name *COBOLY

is directed in the word conversion input section. This generates the word conversion direction section of the conversion rule 99 as shown in Figure 16 by the conversion rule generation means 2. Although this is converted into a form of "item" such as;

& table (relation, *I, *J)
& table (relation, *J, *K)

this has a function to go on with the process only if the relation of;

*J = *I + 1 or *K = *J + 1

is formed.

Figure 20 includes word conversion tables that carry out word conversion according to the word conversion direction section of the conversion rule 99 in Figure 16. In 21b in Figure 21, an example where the origin program 82 in Figure 8 is converted into an intermediate file 21 is shown.

The other origin programs 83, 84, and 85 shown in Figure 8 are written in event driven language (EDL) that is used for the IBM Series/1 computer. To convert these into the target programs 93, 94, and 95 that are written in C language shown in Figure 9, the model statements in the origin language and target language and the word conversion rules such as shown in Figures 12, 13, and 14 must be inputted into the conversion rule input screen.

Conversion rules 99 are generated as shown in Figures 17, 18, and 19 respectively by the conversion rule generation means 2 according to the inputs into the conversion rule input screens in Figures 12, 13, and 14. Figure 20 shows a word conversion table that is used by the word conversion direction section of conversion rule 99 at the time of word conversion.

The origin programs 83, 84, and 85 shown in Figure 8 are first converted into the forms of the intermediate files 21c, 21d, and 21e shown in Figure 21 respectively, and are then converted into the target programs 93, 94, and 95 shown in Figure 9 respectively by the conversion rules 99 shown in Figures 17, 18, and 19 fetched from the conversion rule drive program 100 shown in Figure 22.

Claims

1. A device for converting computer programming languages that converts an origin program written in an origin language, which is one computer language, into a target program written in a target language, which is another computer language, comprising:

   a conversion rule input means for inputting a conversion model expressed by model statements in said origin language and model statements in said target language,

   a conversion rule generation means for generating conversion rules from said inputted model statements,

   an origin program input means for inputting said origin program, and

   a conversion rule drive means for searching for said conversion rules corresponding to said inputted origin program and converting said origin program into said target program according to the content of the searched for conversion rules.

2. A device for converting computer programming languages as claimed in Claim 1 wherein said model statements can be classified into specific key word sections expressing functions and word sections expressing variables in said origin language and said target language.

3. A device for converting computer programming languages as claimed in Claim 2 wherein said conversion rule input means also inputs direc-

tions for conversion between word sections in said origin language and said target language in addition to said model statements.

4. A device for converting computer programming languages as claimed in Claim 3 wherein said conversion rule generation means generates a search key that is used for searching said model statements in said origin language for said conversion rule, generates a target language generation program that is used for generating said target program from said model statements in said target language, generates a word conversion direction section that converts said word sections in said origin language into said word sections in said target language according to said conversion direction, and establishes said conversion rules, taking these as a set.

5. A device for converting computer programming languages as claimed in Claim 4 having a word conversion table and a conversion program for converting said word sections according to the directions of said word conversion direction section.

6. A device for converting computer programming languages as claimed in Claim 5 wherein said origin program input means changes said origin program into an intermediate form that can be easily collated using said conversion rules and wherein said conversion rule drive means searches for said conversion rules based on said intermediate form by means of said search key.

7. A device for converting computer programming languages as claimed in Claim 1 wherein said conversion rule input means, said conversion rule generation means, and said conversion rule drive means do not depend on said origin language and said target language.

8. A device for converting computer programming languages as claimed in Claim 1 wherein a said conversion rule relates a plurality of said model statements of said origin language with said one model statement and converts a plurality of instruction statements of said origin program into one instruction statement of said target program.

9. A device for converting computer programming languages as claimed in Claim 2 having a search key file storing said search keys of said conversion rules and using said search key file for maintenance, such as changes in said conversion rules.

10. A device for converting computer programming languages as claimed in Claim 1 having the content inputted by said conversion rule input means as an independent conversion rule text image file for later maintenance, such as modification of conversion rules.

11. A device for converting computer programming languages that converts an origin program written in an origin language, which is one computer language, into a target program written in a target language, which is another computer language, comprising:

a conversion rule input means having an origin language input section that inputs a certain instruction statement in said origin language using a model statement that is classified into a key word section expressing functions and a word section expressing variables according to said origin language grammar, a target language input section that inputs an instruction statement in said target language corresponding to the function of said certain instruction statement with a model statement that is classified into a key word section expressing functions and a word section expressing variables according to said target language grammar, and a word conversion input section that inputs rules for conversion between said word sections in said origin language and in said target language,

a conversion rule generation means that generates conversion rules that include a section that is generated from the content inputted in said origin language input section and used as a conversion rule search key, a section that is generated from the content inputted in said word conversion input section and used as a word conversion table or a word conversion program call function, and a section that is generated from the content inputted in said target language input section and used as a target language creation program,

a conversion rule database that stores said conversion rules,

an origin language input means that inputs said origin program, and

a conversion rule drive means that fetches said conversion rules corresponding to the inputted instruction statements of said origin program from said conversion rule database by using said conversion rule search key, carries out word conversion by using the fetched word conversion table or word conversion program call function of said conversion rules, creates instruction statements in said target program by using said target language generation program of the fetched conversion rules, and outputs them.

12. A method for converting computer programming languages that converts an origin program writ-

ten in an origin language, which is one computer language, into a target program written in a target language, which is another computer language, comprising the steps of:

inputting a conversion model composed of model statements in said origin language and model statements in said target lauguage,

generating conversion rules from said inputted model statements,

inputting said origin program, and

searching for said conversion rules corresponding to said inputted origin program and converting said origin program into said target program according to the content of the searched for conversion rules.

13. In a step of inputting said conversion model, a method for converting computer programming languages as claimed in Claim 12 wherein said model statements can be classified into specific key word sections expressing functions and word sections expressing variables respectively in said origin language and in said target language.

14. In a step of inputting said conversion model, a method for converting computer programming languages as claimed in Claim 13 wherein a direction for conversion between word sections in said origin language and said target language is also inputted in addition to said model statements.

15. In a step of generating said conversion rules, a method for converting computer programming languages as claimed in Claim 14 wherein a search key is generated that is used for searching said model statements in said origin language for said conversion rules, a target language generation program is generated that is used for generating said target program from said model statements in said target language, a word conversion direction section is generated that converts said word section in said origin language into said word section in said target language according to said conversion directions, and having said conversion rules that take these as sets are established.

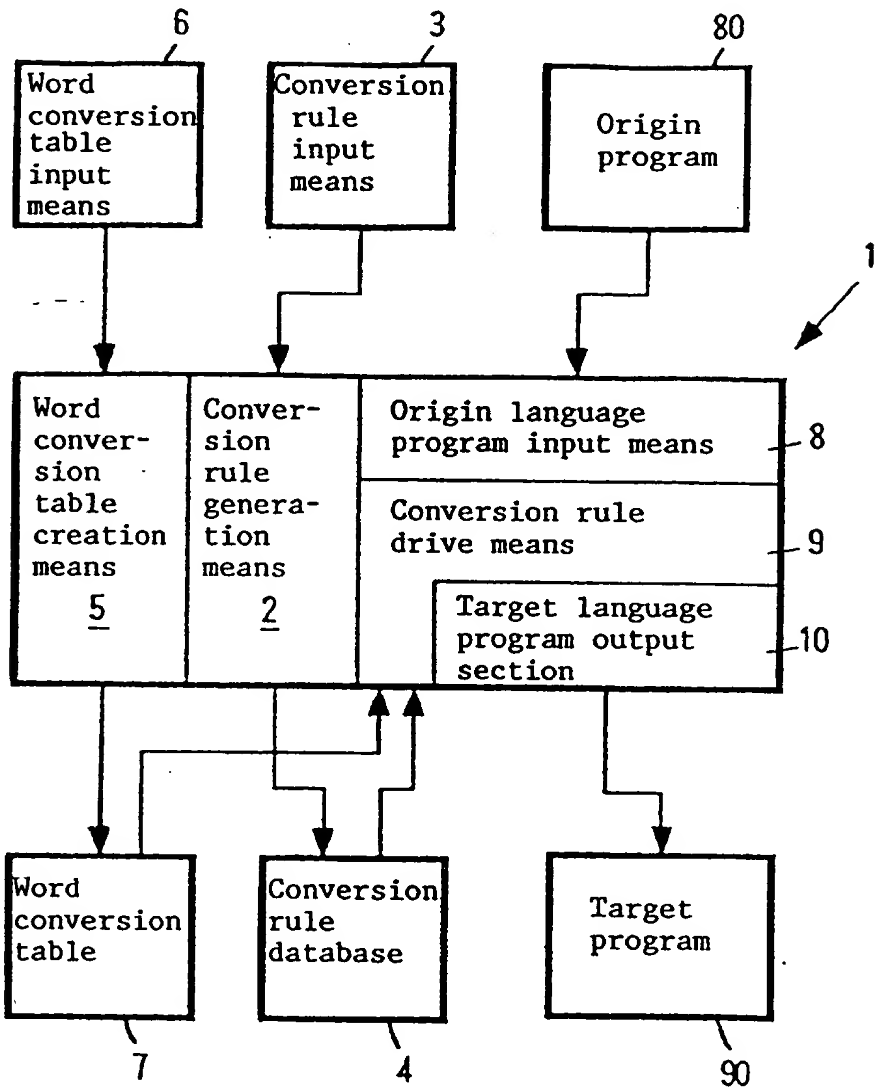

6
Word conversion table input means
3
Conversion rule input means
80
Origin program
1
Word conversion table creation means
5
Conversion rule generation means
2
Origin language program input means
8
Conversion rule drive means
9
Target language program output section
10
Word conversion table
7
Conversion rule database
4
Target program
90

FIG. 1

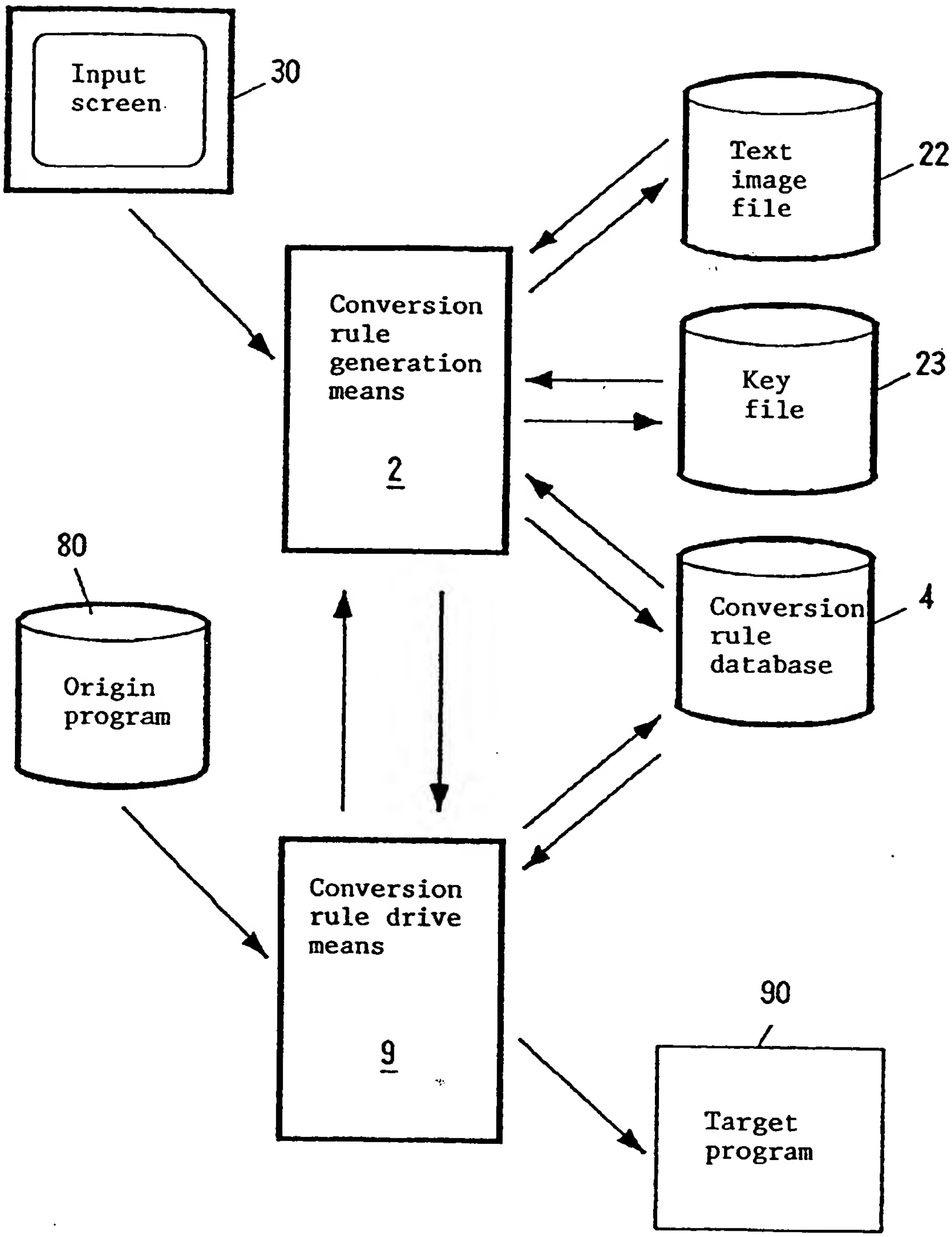

Input
screen
30
Text
image
file
22
Conversion
rule
generation
means
2
Key
file
23
80
Conversion
rule
database
4
Origin
program
Conversion
rule drive
means
9
90
Target
program

FIG. 2

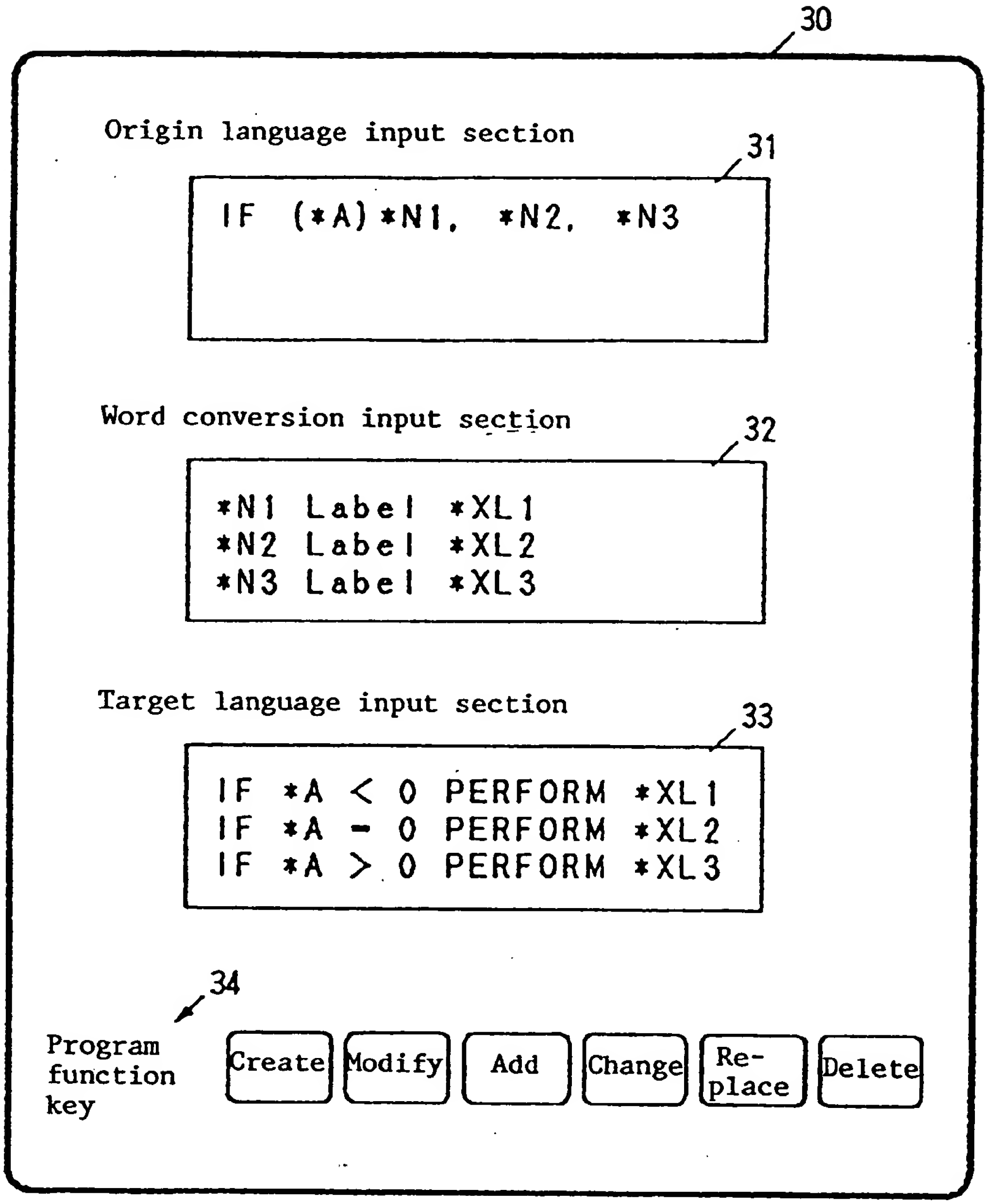

30
Origin language input section
31
IF (*A) *N1, *N2, *N3
Word conversion input section
32
*N1 Label *XL1
*N2 Label *XL2
*N3 Label *XL3
Target language input section
33
IF *A < 0 PERFORM *XL1
IF *A = 0 PERFORM *XL2
IF *A > 0 PERFORM *XL3
34
Program function key
Create
Modify
Add
Change
Re-place
Delete

FIG. 3

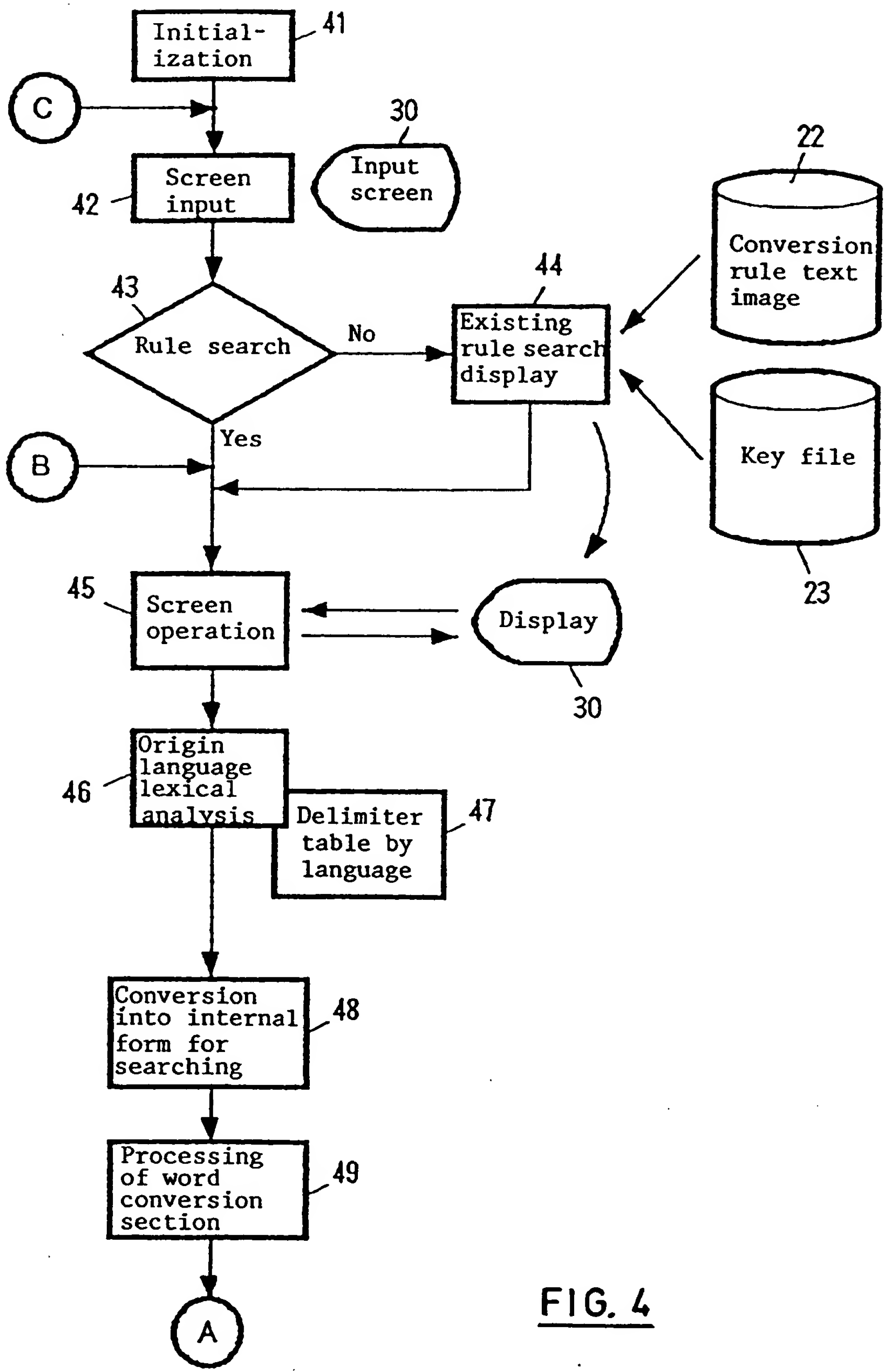

Initial-
ization
41
C
30
Input
screen
Screen
input
42
22
Conversion
rule text
image
43
Rule search
No
44
Existing
rule search
display
Yes
B
Key file
23
45
Screen
operation
Display
30
Origin
language
lexical
analysis
46
Delimiter
table by
language
47
Conversion
into internal
form for
searching
48
Processing
of word
conversion
section
49
A

FIG. 4

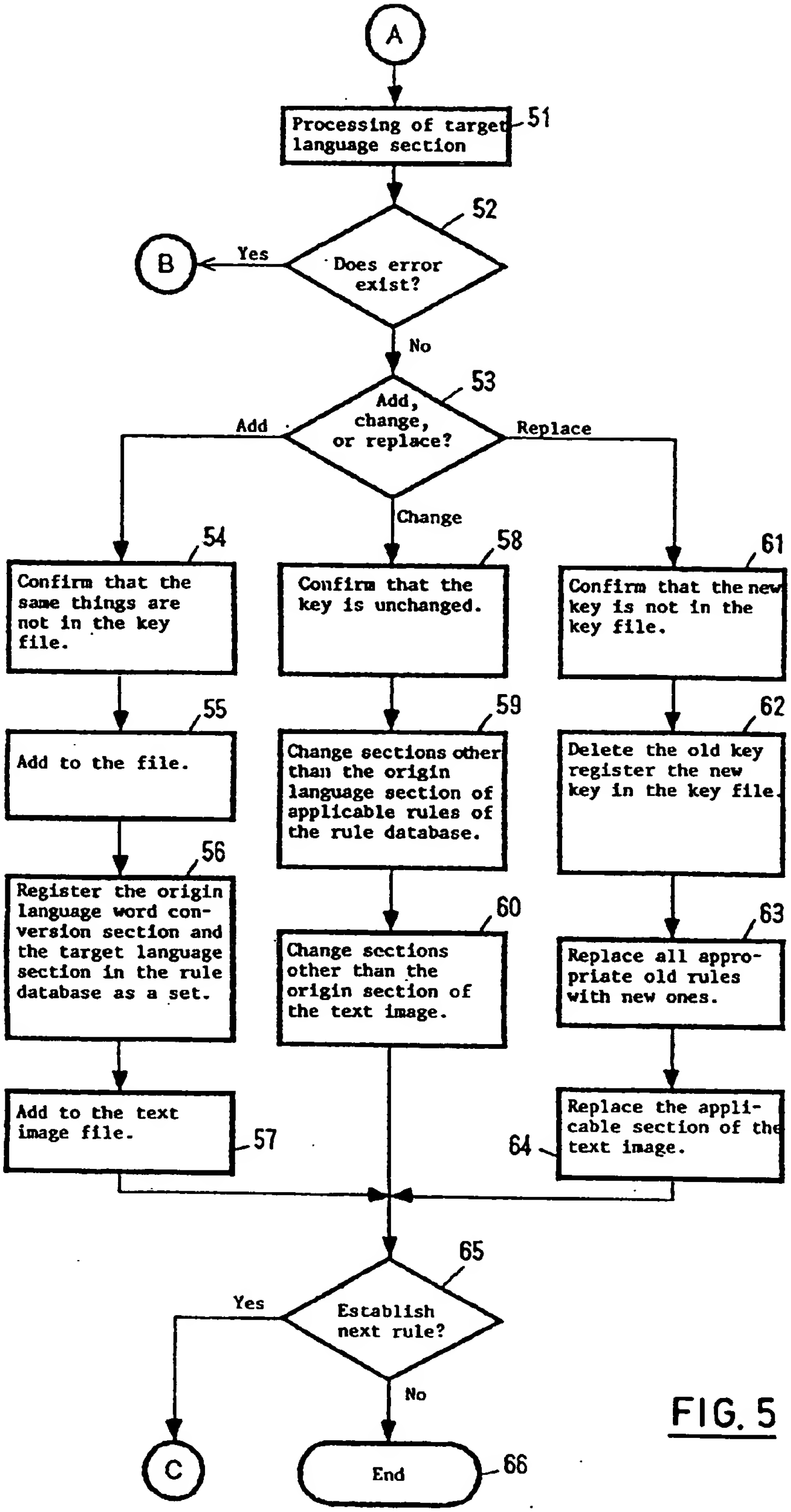

A
Processing of target language section
51
Does error exist?
52
Yes
B
No
Add, change, or replace?
53
Add
Change
Replace
Confirm that the same things are not in the key file.
54
Add to the file.
55
Register the origin language word conversion section and the target language section in the rule database as a set.
56
Add to the text image file.
57
Confirm that the key is unchanged.
58
Change sections other than the origin language section of applicable rules of the rule database.
59
Change sections other than the origin section of the text image.
60
Confirm that the new key is not in the key file.
61
Delete the old key register the new key in the key file.
62
Replace all appropriate old rules with new ones.
63
Replace the applicable section of the text image.
64
Establish next rule?
65
Yes
C
No
End
66

FIG. 5

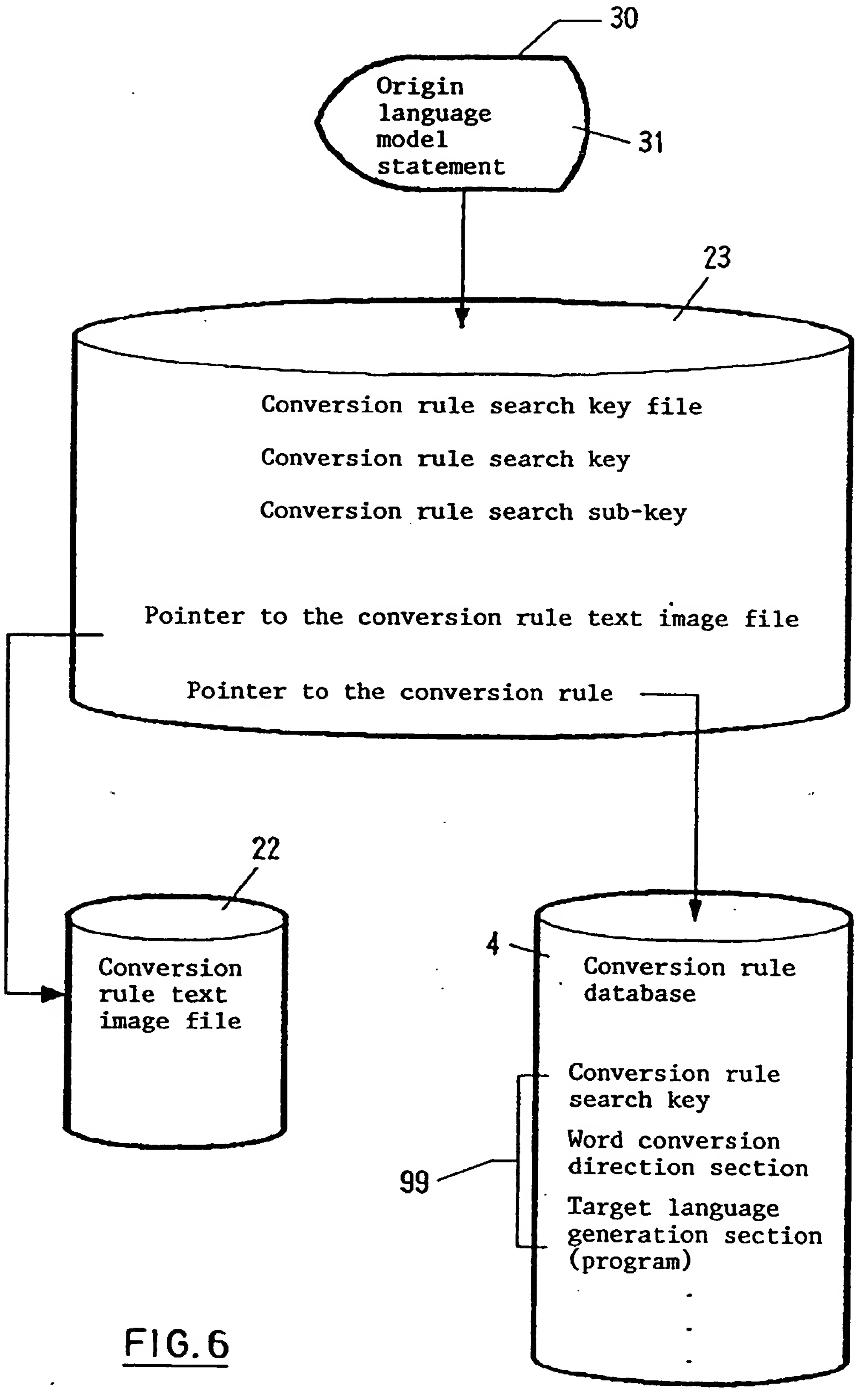
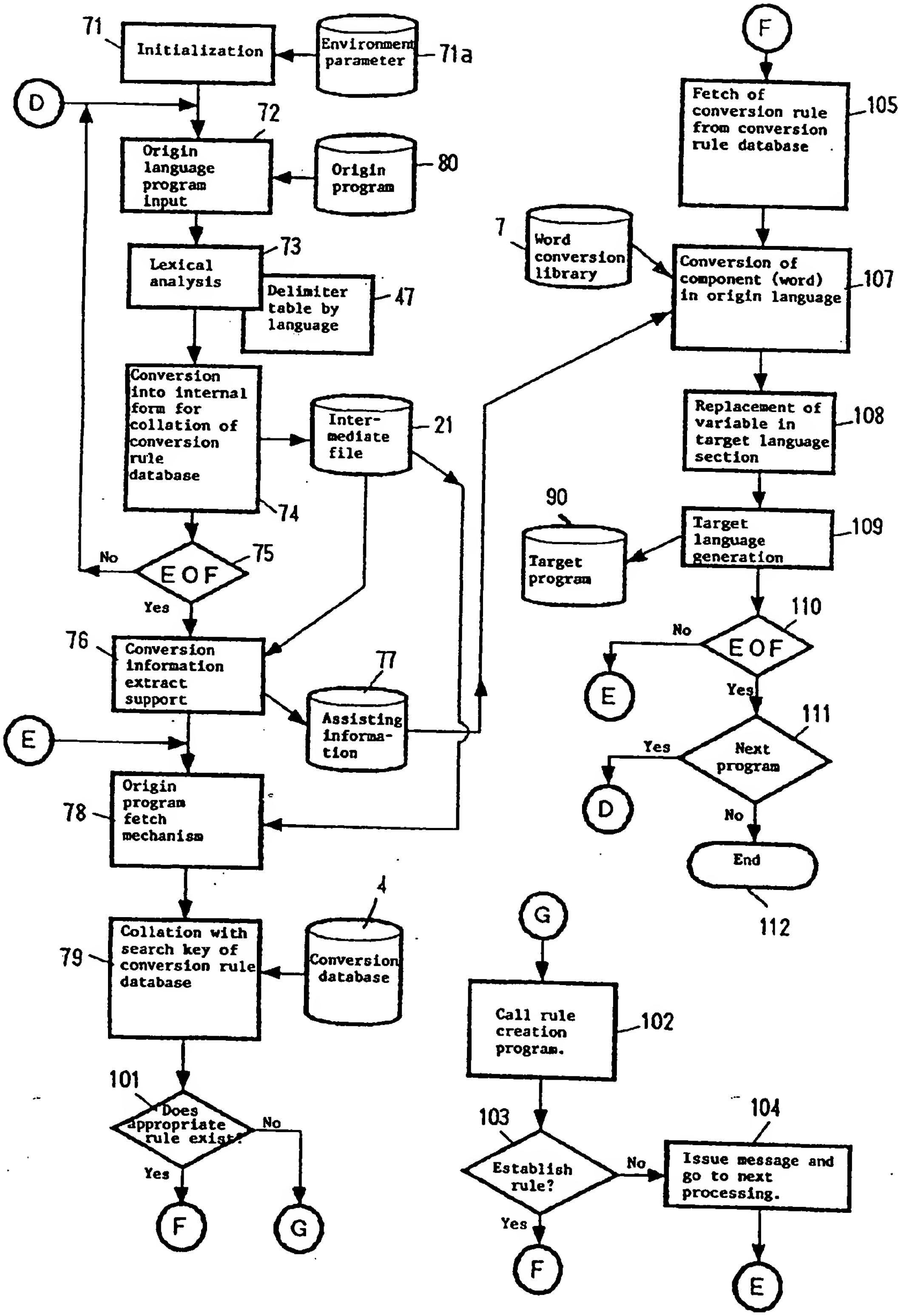

30
Origin language model statement
31
23
Conversion rule search key file
Conversion rule search key
Conversion rule search sub-key
Pointer to the conversion rule text image file
Pointer to the conversion rule
22
Conversion rule text image file
4
Conversion rule database
Conversion rule search key
Word conversion direction section
Target language generation section (program)
99

FIG.6



71
Initialization
Environment parameter
71a
D
72
Origin language program input
Origin program
80
Lexical analysis
73
Delimiter table by language
47
Conversion into internal form for collation of conversion rule database
Inter-mediate file
21
74
No
EOF
75
Yes
76
Conversion information extract support
77
Assisting informa-tion
E
Origin program fetch mechanism
78
Collation with search key of conversion rule database
Conversion database
4
79
101
Does appropriate rule exist?
No
Yes
F
G
F
Fetch of conversion rule from conversion rule database
105
7
Word conversion library
Conversion of component (word) in origin language
107
Replacement of variable in target language section
108
90
Target program
Target language generation
109
110
No
EOF
E
Yes
111
Yes
Next program
D
No
End
112
G
Call rule creation program.
102
103
Establish rule?
No
Issue message and go to next processing.
104
Yes
F
E

FIG. 7

```
IF   (X)   100, 200, 300
TITLE (1)  =  HEAD (1)
TITLE (2)  =  HEAD (2)
TITLE (3)  =  HEAD (3)
MOVE    AREA, (8, #2) , (16, BYTE)
MOVE    AREA, C'  ', (32, BYTE)
MOVE    XAREA, YAREA
```

81 82 FORTRAN 83 84 85 EDL (Event Driven Language)

80

FIG. 8

```
IF X<0 PERFORM L100
IF X=0 PERFORM L200
IF X>0 PERFORM L300
MOVE COBOL-HEAD TO COBOL-TITLE.
memcpy(area, ((char*)reg_2+8), 16);
memset(area, ' ', 32);
xarea=yarea;
```

91 COBOL 92 COBOL 93 94 95 C language

90

FIG. 9

```
IF (*A) *N1, *N2, *N3

    *N1  label  *XL1
    *N2  label  *XL2
    *N3  label  *XL3

  IF *A < 0 PERFORM *XL1.
  IF *A = 0 PERFORM *XL2.
  IF *A > 0 PERFORM *XL3.
```

Origin language input section 31

Word conversion input section 32

Target language input section 33

30

FIG. 10

```
!*TO(*I) = *FROM(*I)
!*TO(*J) = *FROM(*J)
!*TO(*K) = *FROM(*K)

  *I relation *J
  *J relation *K
  *TO name *COBOLX
  *FROM name *COBOLY

  MOVE  *COBOLY TO *COBOLX
```

Origin language input section 31

Word conversion input section 32

Target language input section 33

FIG. 11

```
MOVE *V1, (*V2, *R) , (*L, BYTE)
```

Origin language input section 31

```
*V1 lowercase *V1S
*R  register  *RS
```

Word conversion input section 32

```
memcpy(*V1S, ((char*) *RS + V2), *L) ;
```

Target language input section 33

30

FIG. 12

```
MOVE *V1, C' ', (*L, BYTE)
```

Origin language input section 31

```
*V1 lowercase *V1S
```

Word conversion input section 32

```
memset(*V1S, ' ', *L);
```

Target language input section 33

30

FIG. 13

```
MOVE *V1, *V2                  Origin language input
                               section          31

*V1 lowercase *V1S             Word conversion input
*V2 lowercase *V2S             section          32

*V1S = *V2S;                   Target language input
                               section          33
```

30

FIG. 14

```
                                          Conversion rule
                                          search key
convert (["IF", "(", A, ")", N1, ",",N2, ",",N3])
      <- table(label, N1, XL1)
       & table(label, N2, XL2)            Word conversion
       & table(label, N3, XL3)            direction section
       & Z1 := 'IF '||A||' < 0 PERFORM '||XL1||'.'
       & Z2 := 'IF '||A||' = 0 PERFORM '||XL2||'.'
       & Z3 := 'IF '||A||' > 0 PERFORM '||XL3||'.'
       & prst(Z1, out) & nl (out)
       & prst(Z2, out) & nl (out)         Target language
       & prst(Z3, out) & nl (out)         generation section
                                            (program)
```

99

FIG. 15

Conversion rule search key

```
convert ([ TO ,"(", I, ")","=",FROM, "(", I, ")"],
         [ TO ,"(", J, ")","=",FROM, "(", J, ")"],
         [ TO ,"(", K, ")","=",FROM, "(", K, ")"])
      <- table (relation, I, J)
       & table (relation, J, K)
       & table (name, TO, COBOLX)
       & table (name, FROM, COBOLY)
       & Z1 := 'MOVE '||COBOLY||' TO '||COBOLX||'.'
       & prst(Z1, out)  & nl (out)
```

Word conversion direction section

Target language generation section (program)

99

FIG. 16

Conversion rule search key

```
convert(["MOVE",V1,",","(",V2,",R,")",",","(",L,",","BYTE",")"])
    <- table(lowercase,V1,V1S)
     & table(register,R,RS)
     & Z1 := 'memcpy('||V1S||',((char*)'||RS||'+'||V2||'),
              ||L||');'
     & prst(Z1,out) & nl(out).
```

Word conversion direction section

Target language generation section (program)

FIG. 17

Conversion rule search key

```
convert(["MOVE",V1,",","C' '",",(",L,",",",BYTE",")"])
     <- table(lowercase,V1,V1S)
      & table(register,r,RS)
      & Z1 := 'memset('||V1S||','' '','||L||');'
      & prst(Z1,out) & nl(out).
```

Word conversion direction section

Target language generation section (program)

99

FIG. 18

Conversion rule search key

```
convert(["MOVE",V1,",",V2)
     <- table(lowercase,V1,V1S)
      & table(lowercase,V2,V2S)
      & Z1 := 'memset('||V1S||','' '','||L||');'
      & prst(Z1,out) & nl(out).
```

Word conversion direction section

Target language generation section (program)

99

FIG. 19

```
table(label,N,XL) <- XL := 'L'||N                    % word conversion logic
table(relation,I,J) <- st_to_at(Si,I) & st_to_nb(Si,Ni)
                     & st_to_at(Sj,J) & st_to_nb(Sj,Nj)
                     & Nj  := Ni + 1 .
table(lowercase,X,Y) <- st_to_at(S,X)
                      & Y:= lower(S).
table(name     ,"HEAD" ,"COBOL-HEAD").               % word conversion table
table(name     ,"TITLE","COBOL-TITLE").
table(register ,"#1",reg_1).
table(register ,"#2",reg_2).
```

7

FIG. 20

```
data(1,["IF","(","X",")","100",",","200",",","300"]).  <-- 21a
data(2,["TITLE","(","5",")","=","HEAD","(","5",")"]).  ]
data(3,["TITLE","(","6",")","=","HEAD","(","6",")"]).  ]-21b
data(4,["TITLE","(","7",")","=","HEAD","(","7",")"]).  ]
data(5,["MOVE","AREA",",","(","8",",","#2",")",",",    ]-21c   21d
        "(","16",",","BYTE",")"]).                     ]
data(6,["MOVE","AREA",",","C' '","(","32",",","BYTE",")"]).
data(7,["MOVE","XAREA",",","YAREA"]).  <-- 21e
```

21

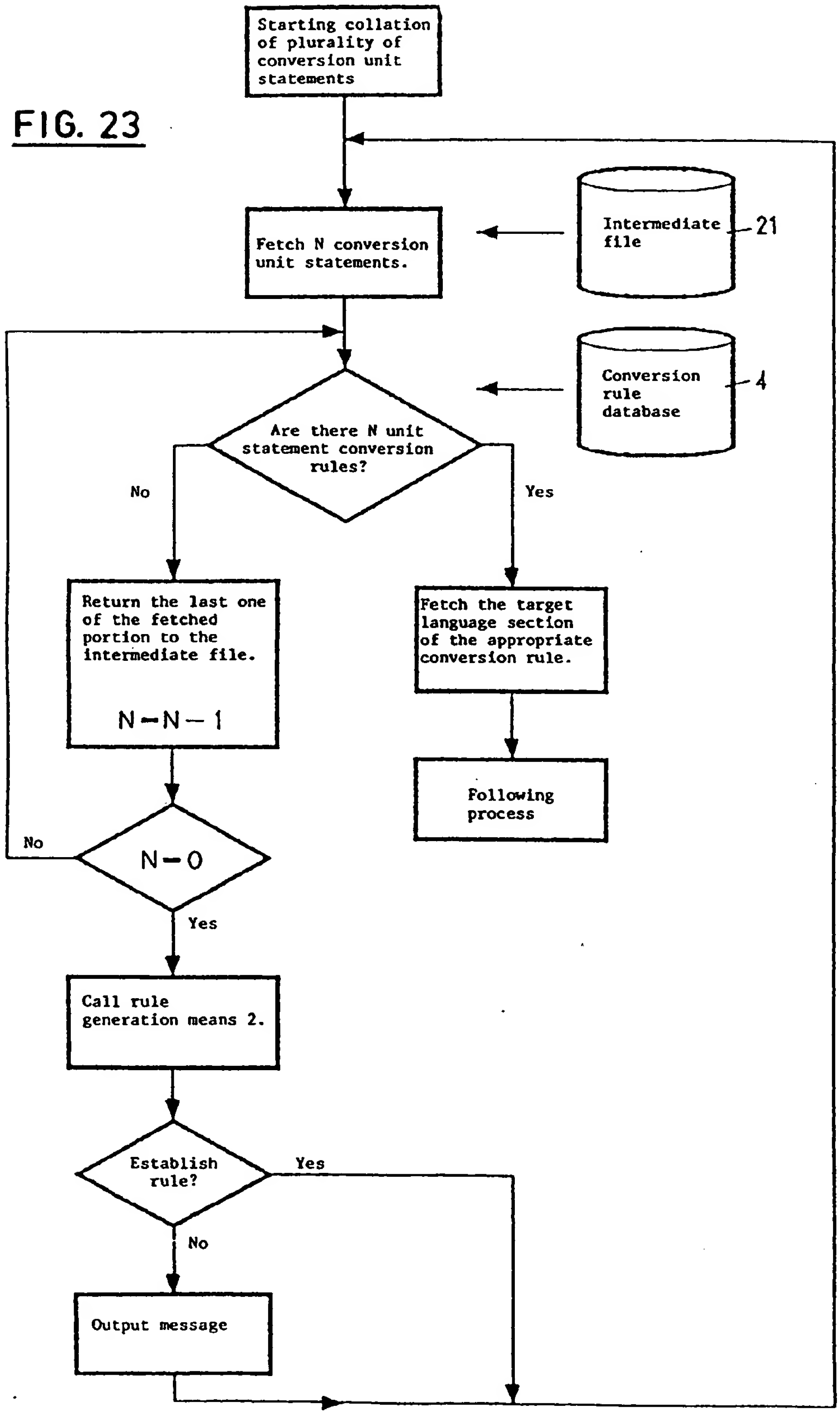
FIG. 21

```
driver
    <- defio(add,out,output,file,'sample.mix.a',[recfm=f,lrecl=80])
     & compute(list,M,data(M,÷),[],List)
     & driver(List)
     & defio(close,out).
driver(M1,M2,M3!List])
    <- data(M1,List1)
     & data(M2,List2)
     & data(M3,List3)
     & convert(List1,List2,List3)
     & driver(List).
driver(M1,M2!List])
    <- data(M1,List1)
     & data(M2,List2)
     & convert(List1,List2)
     & driver(List).
driver(M1!List])
    <- data(M1,List1)
     & convert(List1)
     & driver(List).
driver(M1!List])
    <- data(M,List)
     & rule_gen(List1)
     & driver(List).
driver([]).
```

100

FIG. 22

FIG. 23



Starting collation of plurality of conversion unit statements
Fetch N conversion unit statements.
Intermediate file
21
Conversion rule database
4
Are there N unit statement conversion rules?
No
Yes
Return the last one of the fetched portion to the intermediate file.
N←N−1
Fetch the target language section of the appropriate conversion rule.
Following process
No
N=0
Yes
Call rule generation means 2.
Establish rule?
Yes
No
Output message